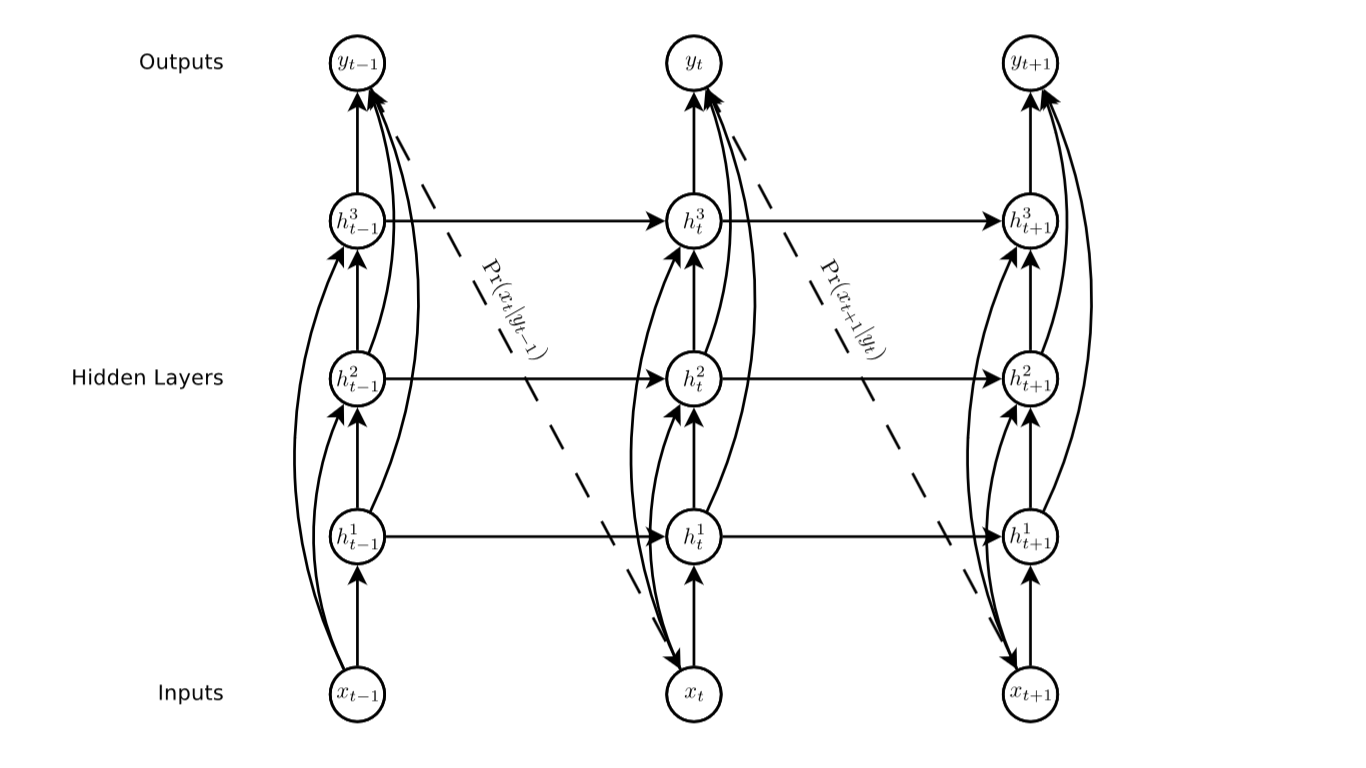
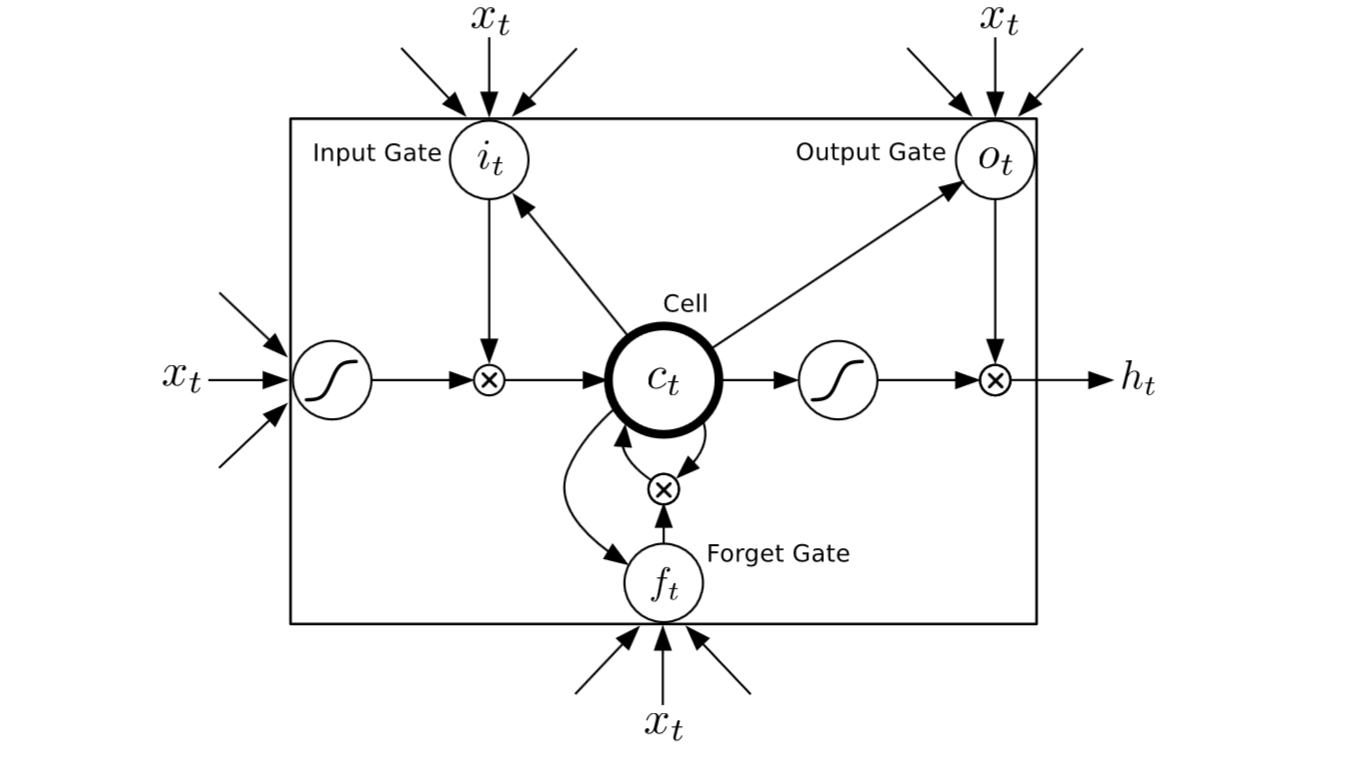
**Generating Sequences With Recurrent Neural Networks**

* RNNs can be trained for sequence generation by processing real data sequences one step at a time and predicting what comes next
* Assuming the predictions are probabilistic, novel sequences can be generated from a trained network by iteratively sampling from the network’s output distribution, then feeding in the sample as input at the next step.
* Long Short-term Memory (LSTM) [16] is an RNN architecture designed to be better at storing and accessing information than standard RNNs. Long Short-term Memory (LSTM) [16] is an RNN architecture designed to be better at storing and accessing information than standard RNNs.



* An input vector sequence x = (x1,...,xT) is passed through weighted connections to a stack of N recurrently connected hidden layers to compute ﬁrst the hidden vector sequences hn = (hn 1,...,hn T) and then the output vector sequence y = (y1,...,yT). Each output vector yt is used to parametrize a predictive distribution Pr(xt+1|yt) over the possible next inputs xt+1. The ﬁrst element x1 of every input sequence is always a null vector whose entries are all zero
* skip connections from the inputs to all hidden layers, and from all hidden layers to the outputs. These make it easier to train deep networks,by reducing the number of processing steps between the bottom of the network and the top, and thereby mitigating the ‘vanishing gradient’ problem
* The output vectors yt are used to parameterise the predictive distribution Pr(xt+1|yt) for the next input. The form of Pr(xt+1|yt) must be chosen carefully to match the input data. In particular, ﬁnding a good predictive distribution for high-dimensional, real-valued data (usually referred to as density modelling), can be very challenging.
* (LSTM) architecture [16], which uses purpose-built memory cells to store information, is better at ﬁnding and exploiting long range dependencies in the data.



* full gradient can be calculated with backpropagation through time. One diﬃculty when training LSTM with the full gradient is that the derivatives sometimes become excessively large leading to numerical problems. To prevent this, all the experiments in this paper clipped the derivative of the loss with respect to the network inputs to the LSTM layers (before the sigmoid and tanh functions are applied) to lie within a predeﬁned range

**Text Prediction**

* Text data is discrete, and is typically presented to neural networks using ‘onehot’ input vectors
* Pr(xt+1|yt) is therefore a multinomial distribution, which can be naturally parameterised by a softmax function at the output layer
* In the case of softmax models, diﬃculty is the high computational cost of evaluating all the exponentials during training.
* Furthermore, word-level models are not applicable to text data containing non-word strings, such as multi-digit numbers or web addresses.
* Character-level language modelling with neural networks has recently been considered , and found to give slightly worse performance than equivalent word-level models.

**Penn Treebank Experiments:**

* The ﬁrst set of text prediction experiments focused on the **Penn Treebank portion of the Wall Street Journal corpus,** main purpose was to gauge the predictive power of the network, rather than to generate interesting sequences.
* The experiments compared the performance of word and character-level LSTM predictors on the Penn corpus. In both cases, the network architecture was a single hidden layer with 1000 LSTM units. For the character-level network the input and output layers were size 49, giving approximately 4.3M weights in total, while the word-level network had 10,000 inputs and outputs and around 54M weights
* All networks were trained with stochastic gradient descent, using a learn rate of 0.0001 and a momentum of 0.99. The LSTM derivatives were clipped in the range [−1,1]
* For prediction problems, where the inputs are the targets, it is legitimate to allow the network to adapt its weights as it is being evaluated (so long as it only sees the test data once). Mikolov refers to this as dynamic evaluation.
* We also experiment with two types of regularization: weight noise with a std. deviation of 0.075 applied to the network weights at the start of each training sequence, and adaptive weight noise, where the variance of the noise is learned along with the weights using a Minimum description Length (or equivalently, variational inference) loss function.
* One advantage of adaptive weight is that early stopping is not needed (the network can safely be stopped at the point of minimum total ‘description length’ on the training data).
* The results are presented with two equivalent metrics: bits-per-character (BPC), which is the average value of −log2 Pr(xt+1|yt) over the whole test set; and perplexity which is two to the power of the average number of bits per word (the average word length on the test set is about 5.6 characters, so perplexity ≈ 2^5.6BPC). Perplexity is the usual performance measure for language modeling.
* LSTM is better at rapidly adapting to new data than ordinary RNNs.

**Wikipedia Experiments:**

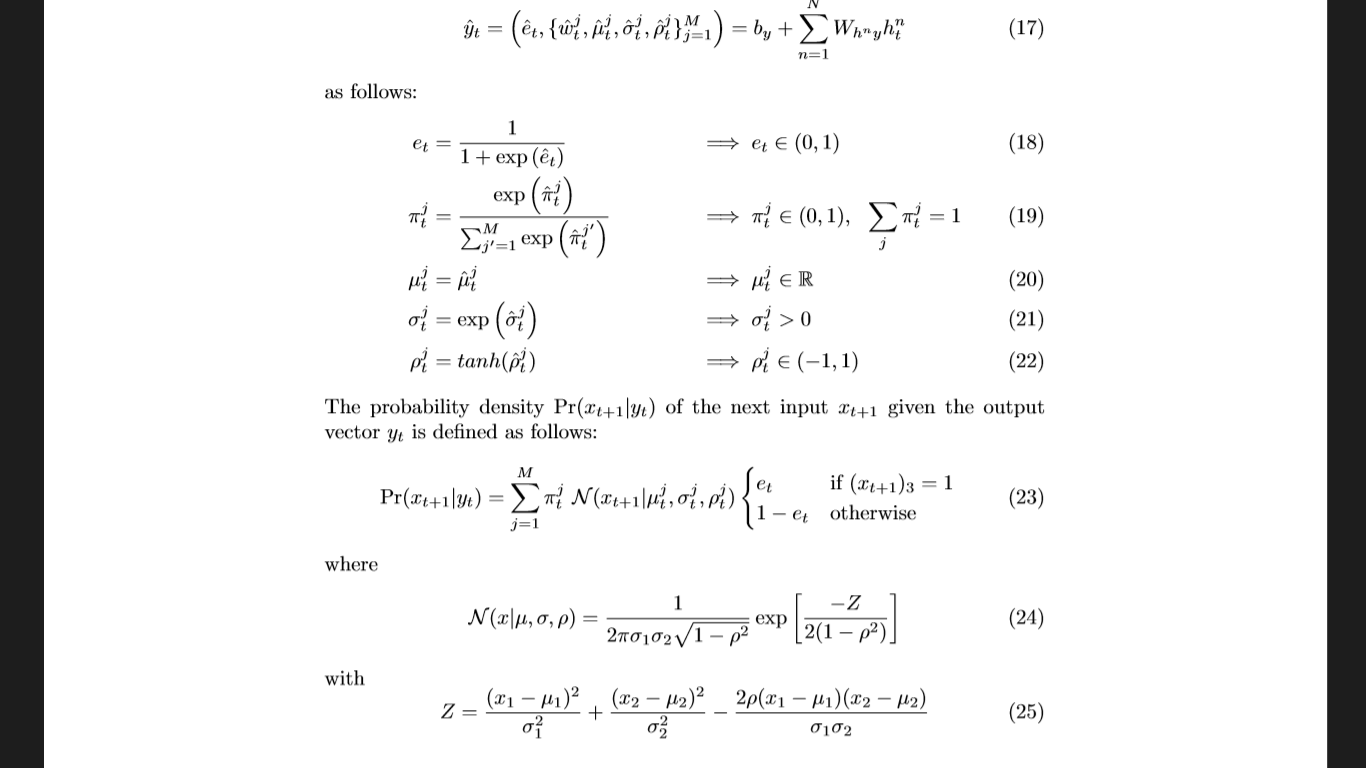
* Hutter prize [17]: to compress the ﬁrst 100 million bytes of the complete English Wikipedia data, The ﬁle had to include not only the compressed data, but also the code implementing the compression algorithm. Its size can therefore be considered a measure of the minimum description length [13] of the data using a two part coding scheme
* The ﬁrst 96M bytes in the data were evenly split into sequences of 100 bytes and used to train the network, with the remaining 4M were used for validation. The data contains a total of 205 one-byte unicode symbols.
* the network predicted a single byte at a time, and therefore had size 205 input and output layers.
* networks internal state (that is, the output activations ht of the hidden layers, and the activations ct of the LSTM cells within the layers) were only reset every 100 sequences
* The order of the sequences was not shuﬄed during training.The network was therefore able to access information from up to 10K characters in the past when making predictions.
* The error terms were only back propagated to the start of each 100 byte sequence, meaning that the gradient calculation was approximate. This form of truncated backpropagation has been considered before for RNN language modelling [23], and found to speed up training (by reducing the sequence length and hence increasing the frequency of stochastic weight updates) without aﬀecting the network’s ability to learn long-range dependencies.
* A much larger network was used for this data with seven hidden layers of 700 LSTM cells, giving approximately 21.3M weights. The network was trained with stochastic gradient descent, using a learn rate of 0.0001 and a momentum of 0.9. It took four training epochs to converge. The LSTM derivatives were clipped in the range [−1,1].
* we tested the network on the validation data with and without dynamic evaluation
* The model understood the context,also generated other language words,put heading in start and see also section in end,learning correct punctuations.But it is not very meaningful except short phrases

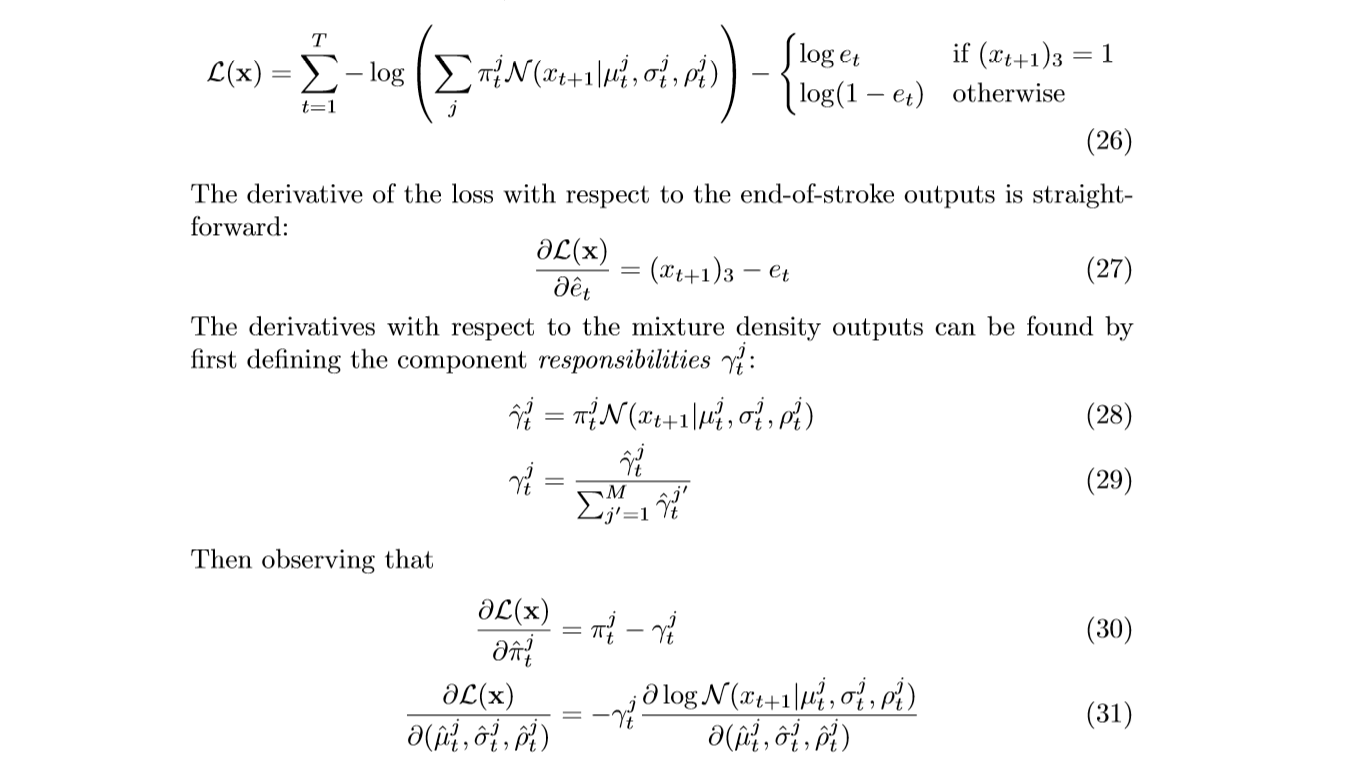
**Handwriting Prediction:**

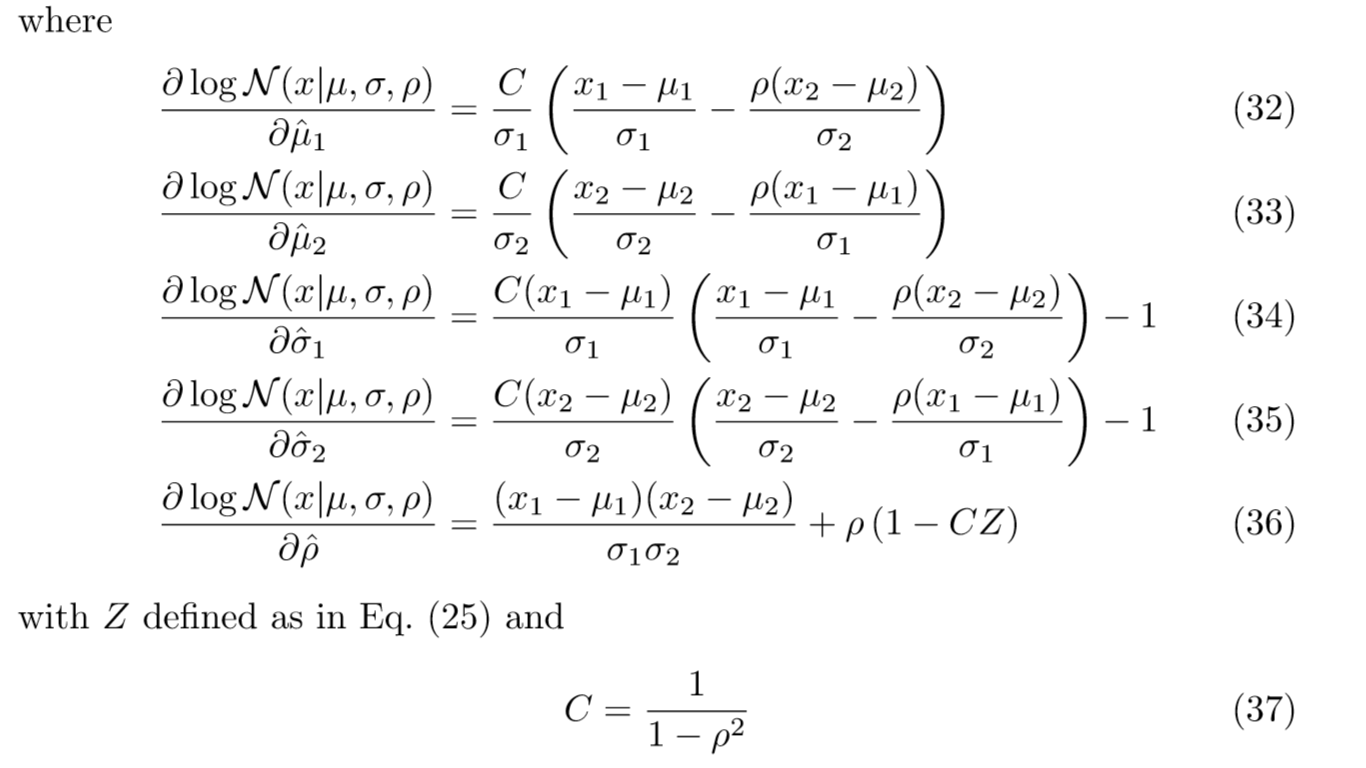
* To generate convincing real-valued sequences, we applied the network to online handwriting data (online in this context means that the writing is recorded as a sequence of pen-tip locations, as opposed to oﬄine handwriting, where only the page images are available).
* IAM online handwriting database (IAM-OnDB) : IAM-OnDB consists of handwritten lines collected from 221 diﬀerent writers using a ‘smart whiteboard’. The writers were asked to write forms from the Lancaster-Oslo-Bergen text corpus [19], and the position of their pen was tracked using an infra-red device in the corner of the board.
* The original input data consists of the x and y pen co-ordinates and the points in the sequence when the pen is lifted oﬀ the whiteboard.
* The network was trained to predict the x,y coordinates and the end of-stroke markers one point at a time.
* Predicting the pen traces one point at a time gives the network maximum ﬂexibility to invent novel handwriting, but also requires a lot of memory, with the average letter occupying more than 25 timesteps and the average line occupying around 700.Predicting delayed strokes is especially demanding
* IAM-OnDB is divided into a training set, two validation sets and a test set, containing respectively 5364, 1438, 1518 and 3859 handwritten lines taken from 775, 192, 216 and 544 forms.
* The principal challenge in applying the prediction network to online handwriting data was determining a predictive distribution suitable for real-valued inputs

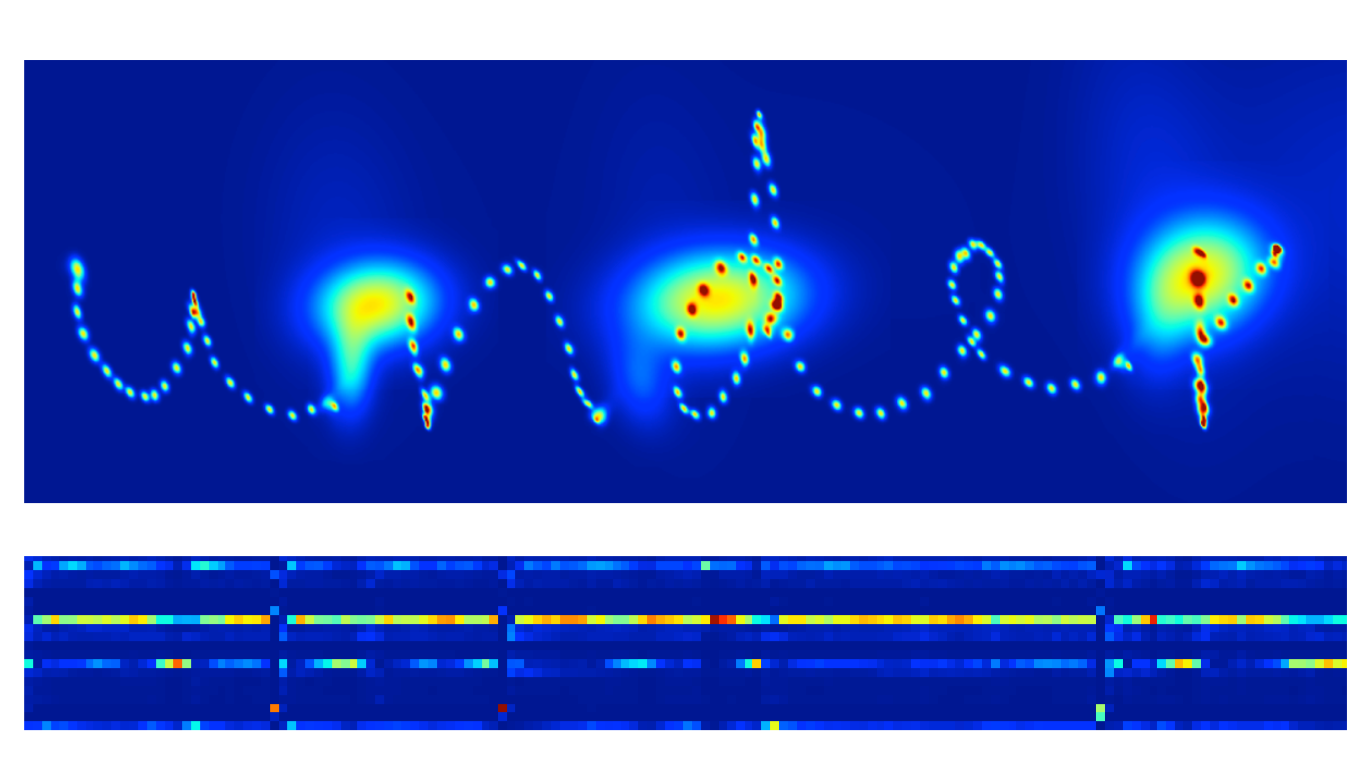
**Mixture Density Outputs:**

* The idea of mixture density networks [2, 3] is to use the outputs of a neural network to parameterise a mixture distribution. A subset of the outputs are used to deﬁne the mixture weights, while the remaining outputs are used to parameterise the individual mixture components.
* Each input vector xt consists of a real-valued pair x1,x2 that deﬁnes the pen oﬀset from the previous input, along with a binary x3 that has value 1 if the vector ends a stroke (that is, if the pen was lifted oﬀ the board before the next vector was recorded) and value 0 otherwise. A mixture of bivariate Gaussians was used to predict x1 and x2, while a Bernoulli distribution was used for x3. Each output vector yt therefore consists of the end of stroke probability e, along with a set of means µj, standard deviations σj, correlations ρj and mixture weights πj for the M mixture components
* The mean and standard deviation are two dimensional vectors, whereas the component weight, correlation and end-of-stroke probability are scalar.

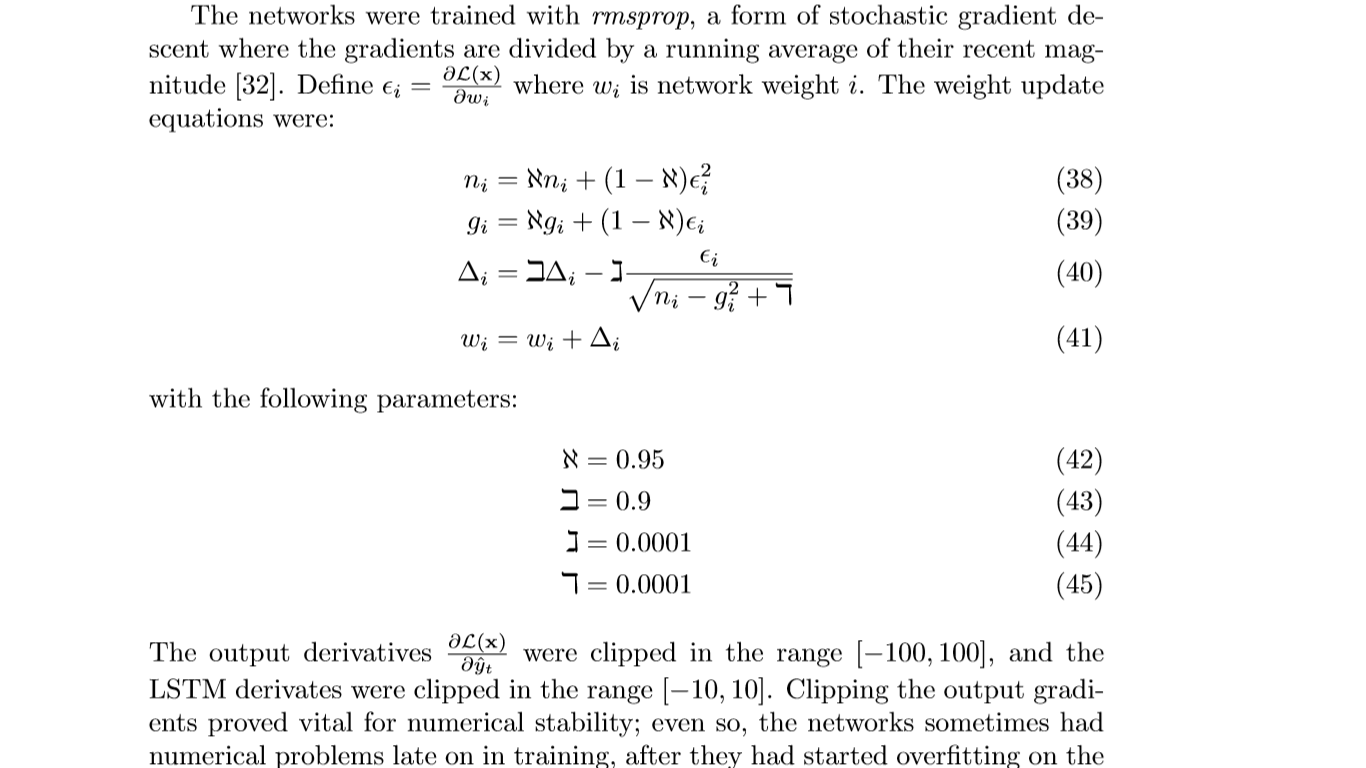






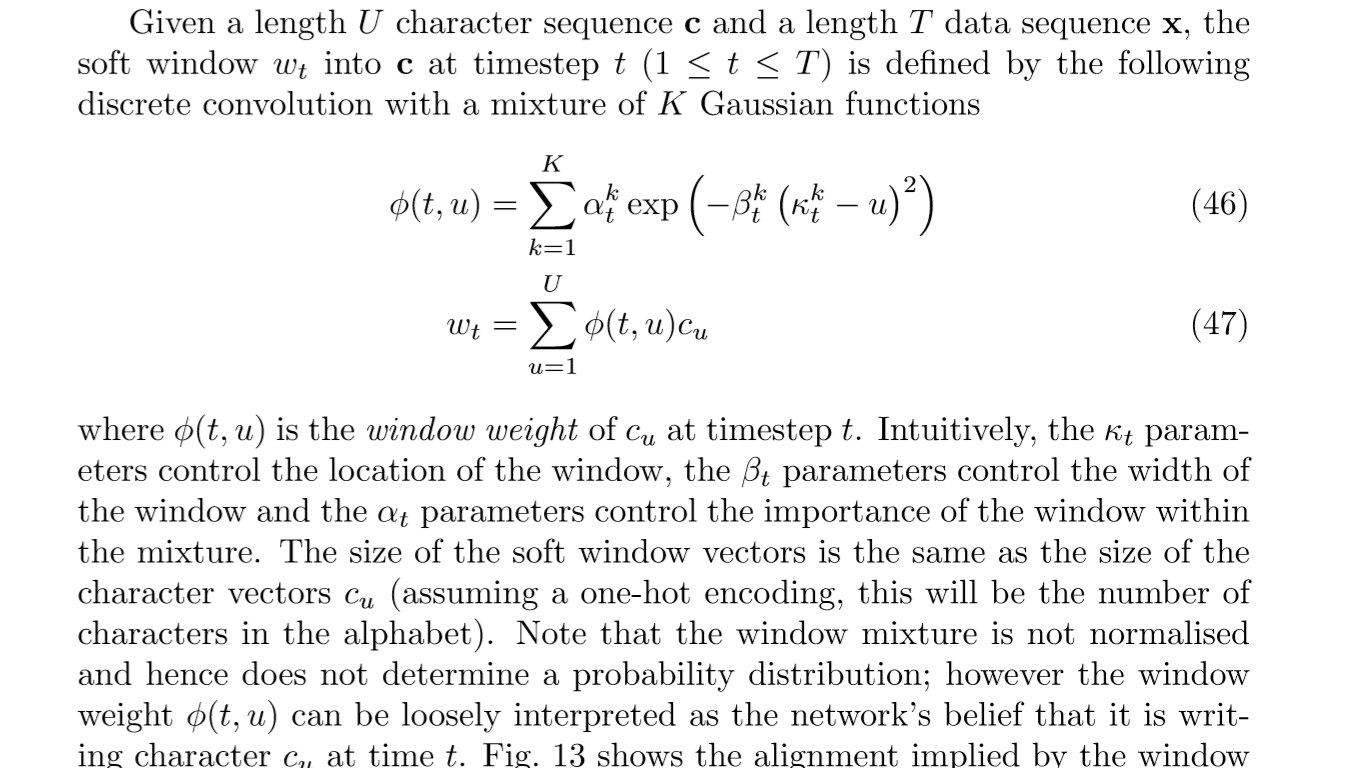


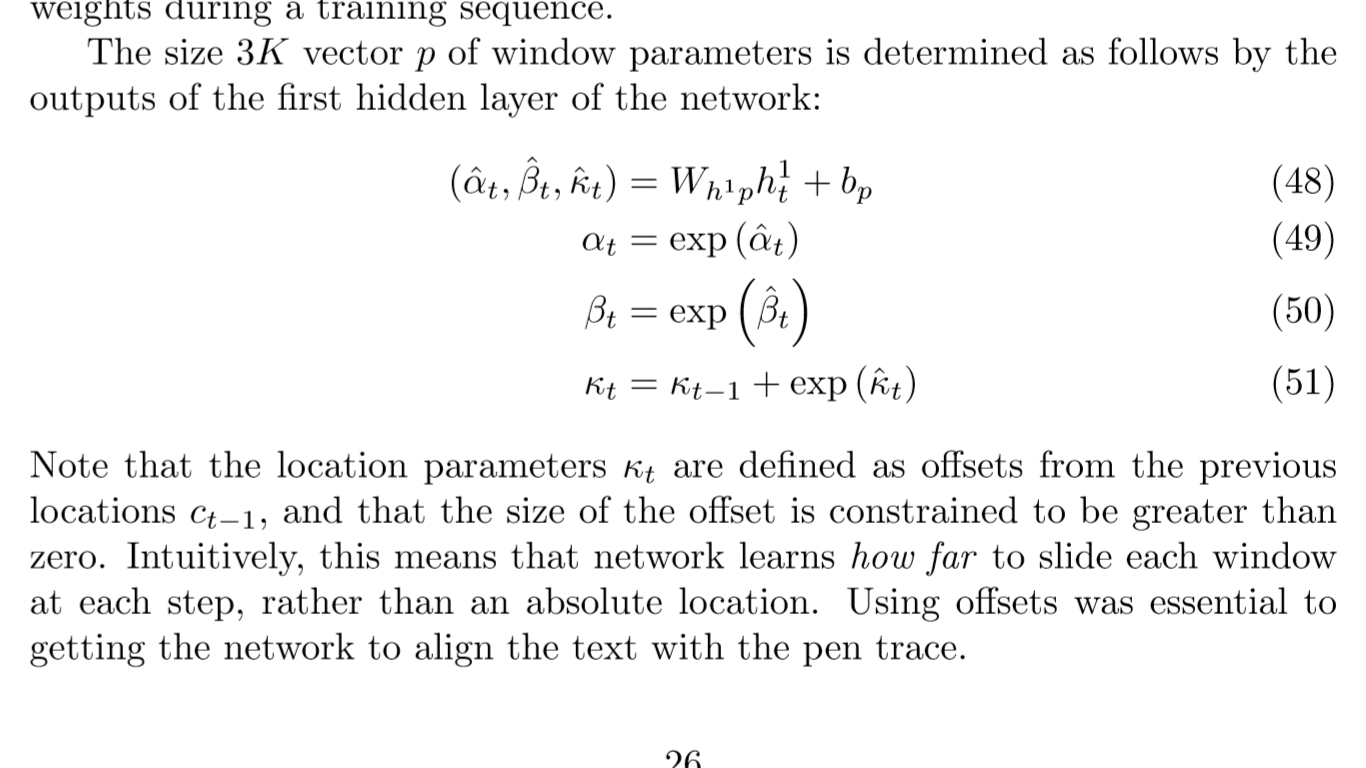
* The top heatmap shows the sequence of probability distributions for the predicted pen locations as the word ‘under’ is written. The densities for successive predictions are added together, giving high values where the distributions overlap.
* Two types of prediction are visible from the density map: the small blobs that spell out the letters are the predictions as the strokes are being written, the three large blobs are the predictions at the ends of the strokes for the ﬁrst point in the next stroke having much higher variance
* The bottom heatmap shows the mixture component weights during the same sequence. The stroke ends are also visible here, with the most active components switching oﬀ in three places, and other components switching on: evidently end-of-stroke predictions use a diﬀerent set of mixture components from in-stroke predictions.
* The co-ordinate oﬀsets were normalised to mean 0, std. dev. 1 over the training set. 20 mixture components were used to model the oﬀsets, giving a total of 120 mixture parameters per timestep (20 weights, 40 means, 40 standard deviations and 20 correlations). A further parameter was used to model the end-of-stroke probability, giving an output layer of size 121.
* Two network architectures were compared for the hidden layers: one with three hidden layers, each consisting of 400 LSTM cells, and one with a single hidden layer of 900 LSTM cells. Both networks had around 3.4M weights. The three layer network was retrained with adaptive weight noise [8], with all std. devs. initialised to 0.075. Training with ﬁxed variance weight noise proved ineﬀective, probably because it prevented the mixture density layer from using precisely speciﬁed weights

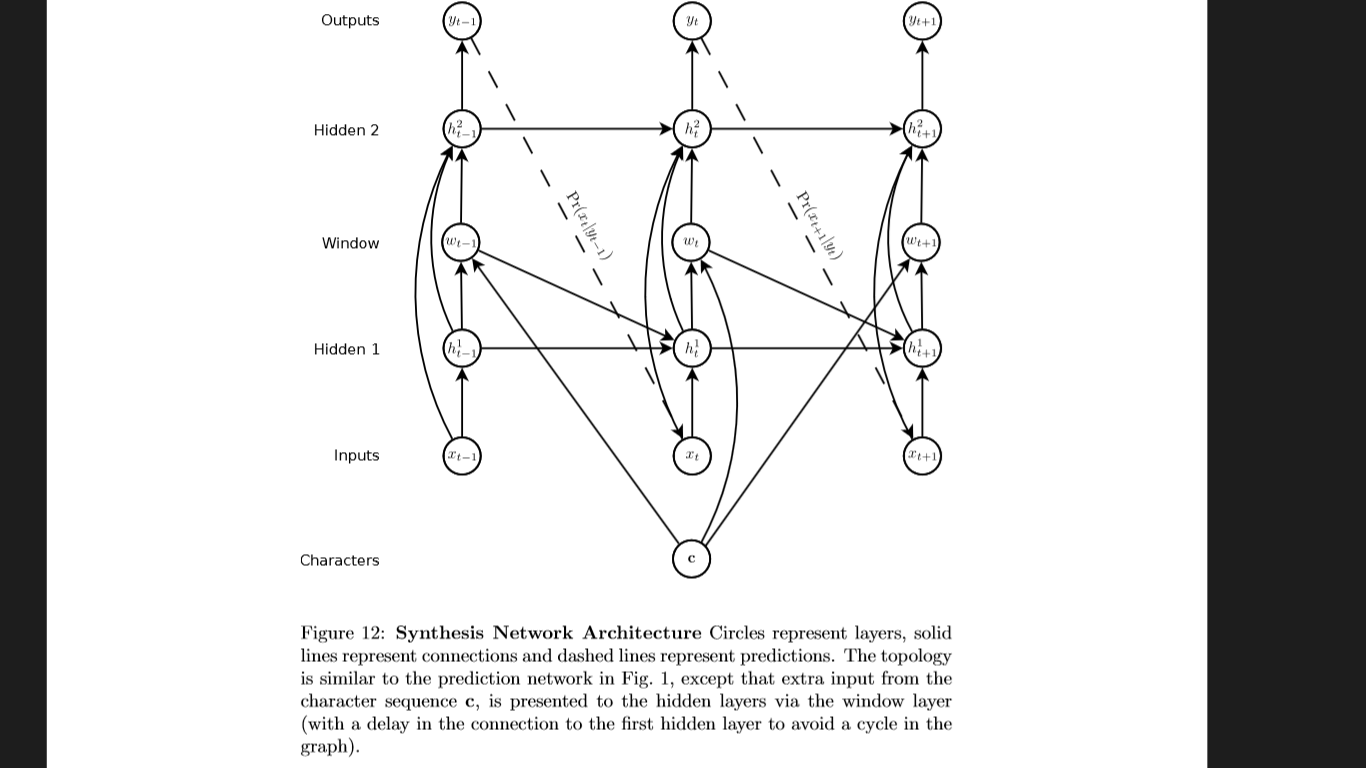


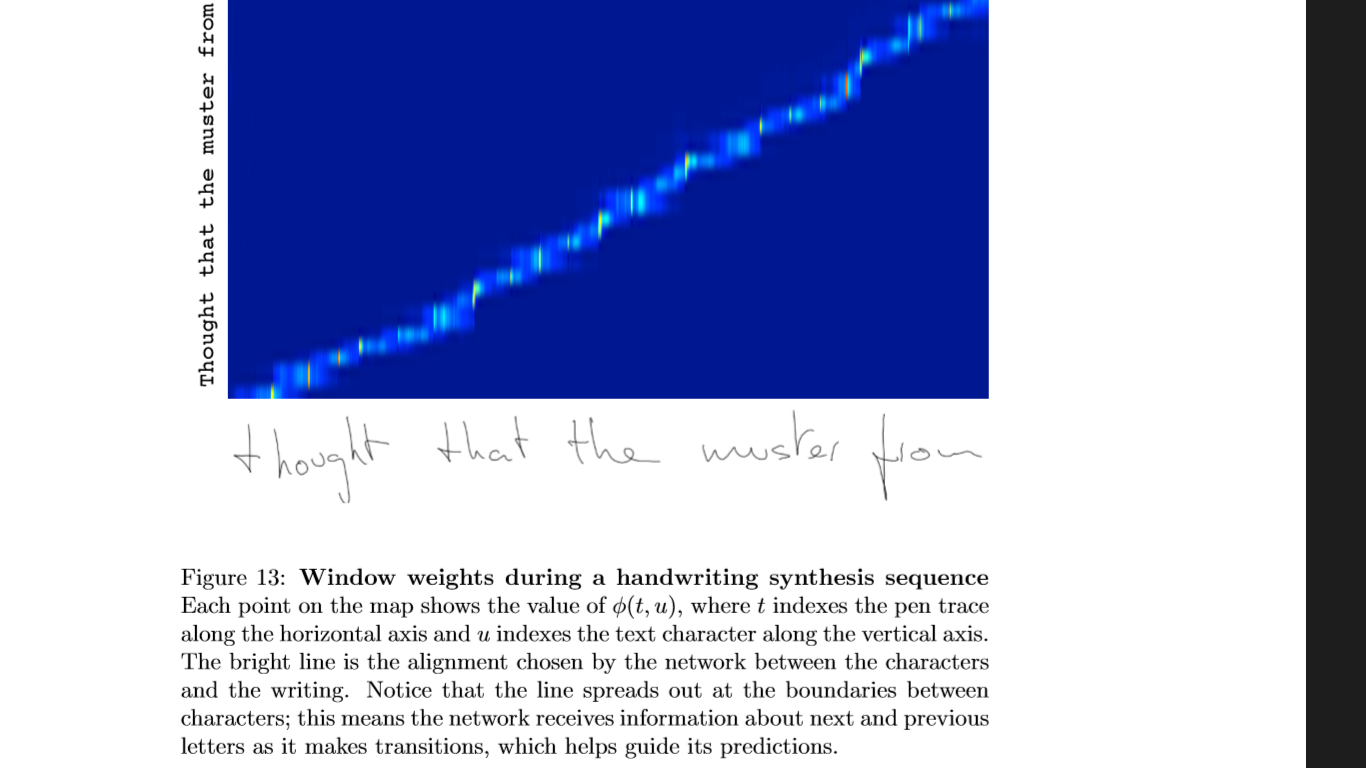
**Handwriting Synthesis:**

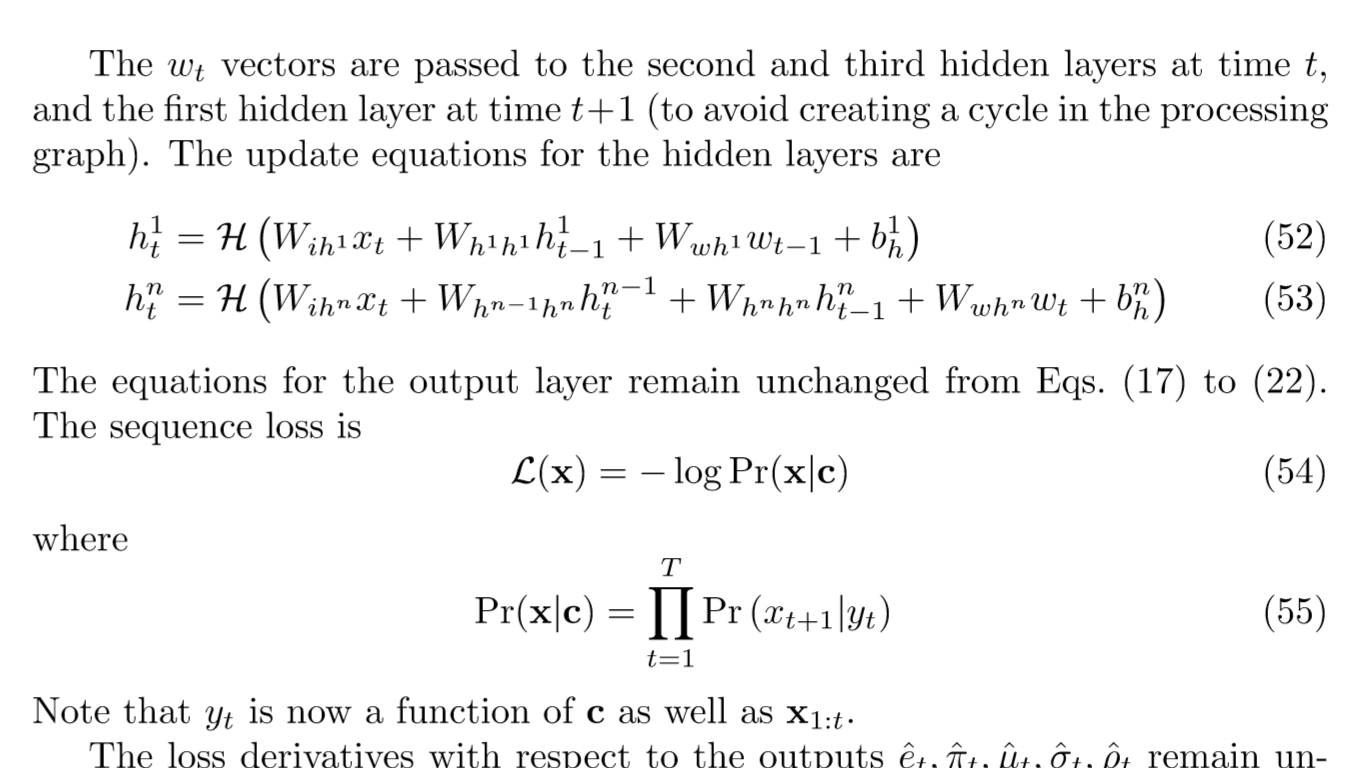
* An augmentation allows a prediction network to generate data sequences conditioned on some high-level annotation sequence (a character string, in the case of handwriting synthesis)
* The main challenge in conditioning the predictions on the text is that the two sequences are of very diﬀerent lengths (the pen trace being on average twenty ﬁve times as long as the text), and the alignment between them is unknown until the data is generated
* This is because the number of co-ordinates used to write each character varies greatly according to style, size, pen speed etc.
* RNN transducer is incapable of doing this
* A ‘soft window’ is convolved with the text string and fed in as an extra input to the prediction network.
* The parameters of the window are output by the network at the same time as it makes the predictions, so that it dynamically determines an alignment between the text and the pen locations. Put simply, it learns to decide which character to write next.

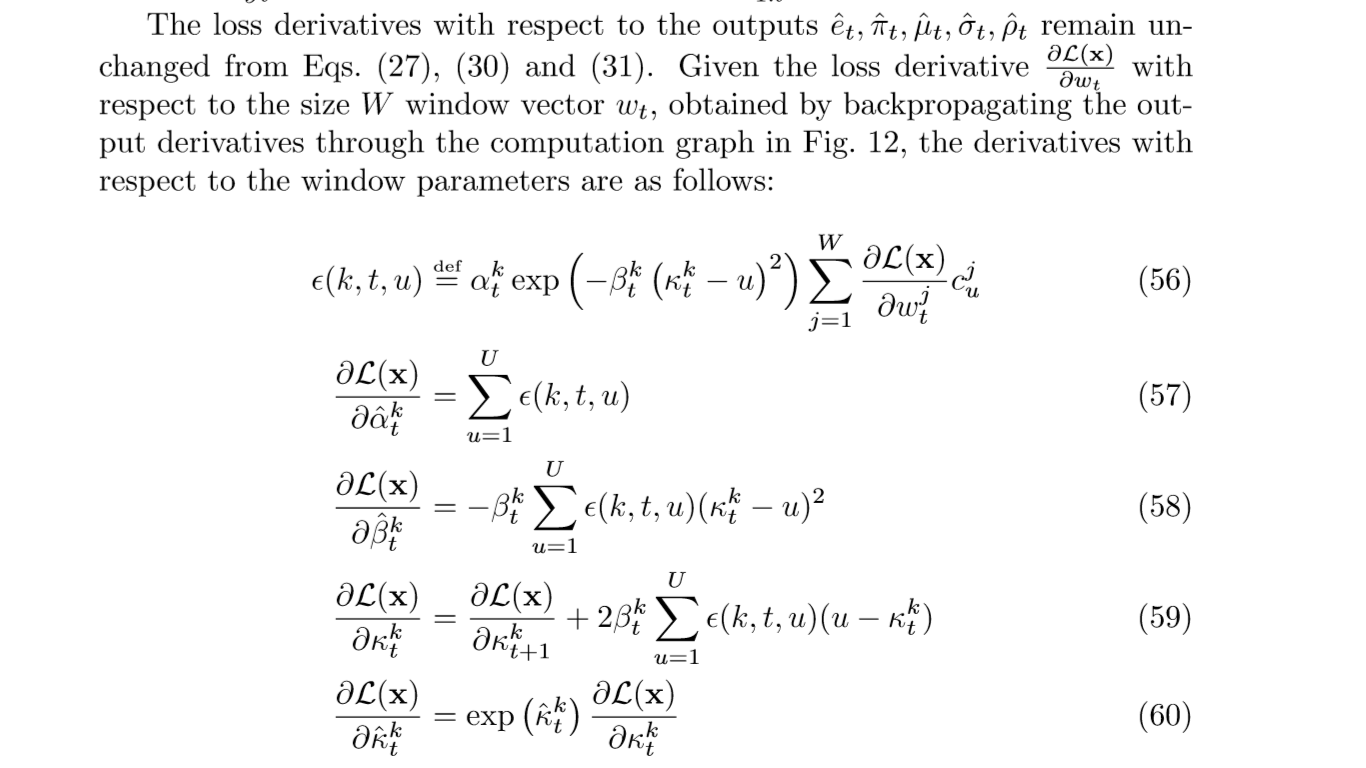








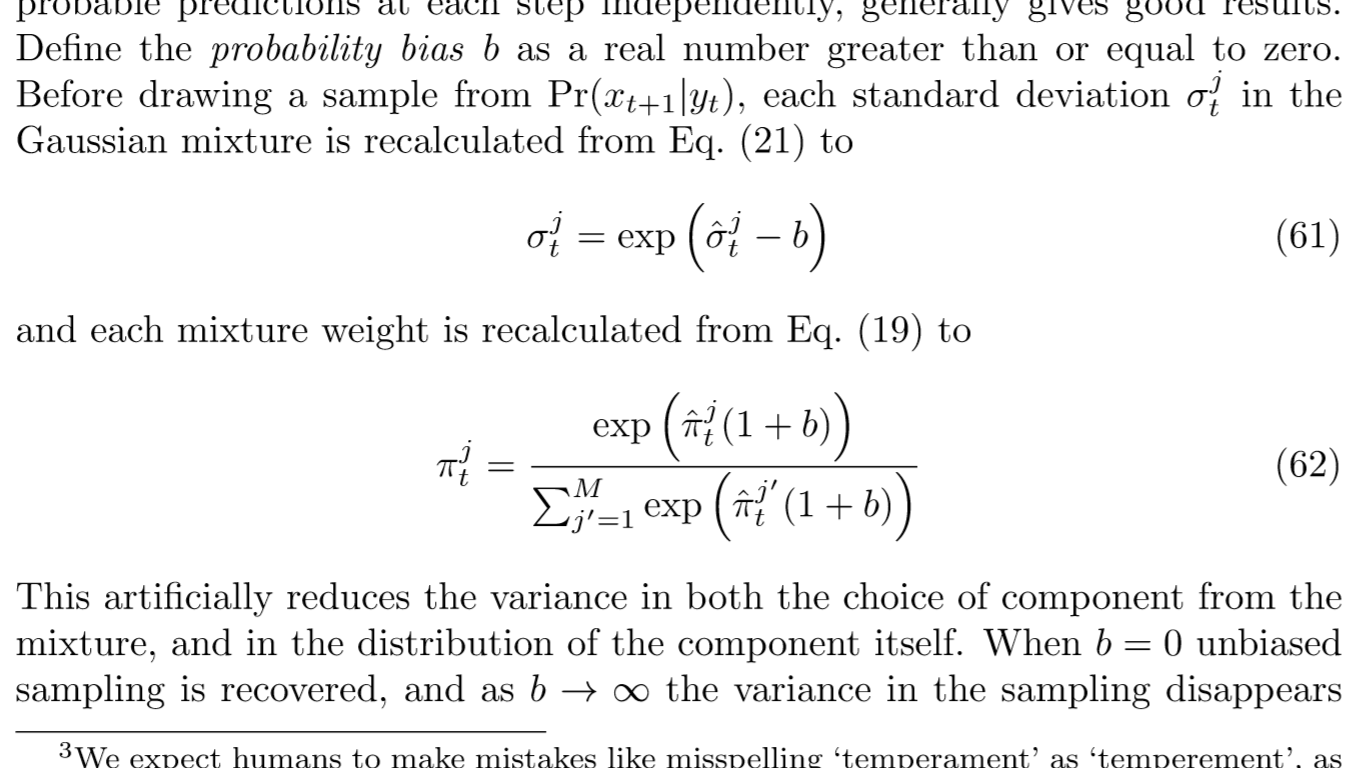




* The synthesis network was applied to the same input data as the handwriting prediction network. The character-level transcriptions from the IAM-OnDB were now used to deﬁne the character sequences c. The full transcriptions contain 80 distinct characters However we used only a subset of 57,digits and most of the punctuation characters replaced with a generic ‘nonletter’ label
* The network architecture was: three hidden layers of 400 LSTM cells each, 20 bivariate Gaussian mixture components at the output layer and a size 3 input layer. The character sequence was encoded with one-hot vectors, and hence the window vectors were size 57. A mixture of 10 Gaussian functions was used for the window parameters, requiring a size 30 parameter vector. The total number of weights was increased to approximately 3.7M
* Given c, an unbiased sample can be picked from Pr(x|c) by iteratively drawing xt+1 from Pr(xt+1|yt),we must also decide when the synthesis network has ﬁnished writing the text and should stop making any future decisions. To do this, we use the following heuristic: as soon as φ(t,U + 1) > φ(t,u) ∀ 1 ≤ u ≤ U the current input xt is deﬁned as the end of the sequence and sampling ends.
* stylistic traits, such as character size, slant, cursiveness etc. vary widely between the samples, but remain more-or-less consistent within them. This suggests that the network identiﬁes the traits early on in the sequence, then remembers them until the end
* The number of mistakes increases when less common words or phrases are included in the character sequence. Presumably this is because the network learns an implicit character-level language model from the training set that gets confused when rare or unknown transitions occur.

**BIASED SAMPLING:**

* Intuitively, we would expect the network to give higher probability to good handwriting because it tends to be smoother and more predictable than bad handwriting
* a simple heuristic, where the sampler is biased towards more probable predictions at each step independently, generally gives good results



As the bias increases the diversity decreases and the samples tend towards a kind of ‘average handwriting’ which is extremely regular and easy to read

**PRIMED SAMPLING:**

* Another reason to constrain the sampling would be to generate handwriting in the style of a particular writer
* it is possible to mimic a particular style by ‘priming’ the network with a real sequence, then generating an extension with the real sequence still in the network’s memory. This can be achieved for a real x, c and a synthesis character string s by setting the character sequence to c0 = c + s and clamping the data inputs to x for the ﬁrst T timesteps, then sampling as usual until the sequence ends.
* Primed sampling and reduced variance sampling can also be combined this tends to produce samples in a ‘cleaned up’ version of the priming style, with overall stylistic traits such as slant and cursiveness retained, but the strokes appearing smoother and more regular